REMARKS

Claims 1-28 are currently pending. By this amendment, Applicants add Claim 28. Authorization is hereby given to charge Deposit Account 50-0629 in the amount of $102.00 for the addition of an independent claim in excess of three and a claim in excess of 20 total claims.

The Examiner has rejected Claims 1-3, 5-6 and 8-16 as unpatentable over Matheny in view of Selby; Claims 4 and 7 as unpatentable over Matheny in view of Selby and Andrew; Claims 17-24 as unpatentable over Andrew in view of Matheny; and Claims 25-27 as unpatentable over Andrew in view of Matheny and Selby. For the reasons set forth below, Applicants respectfully submit that the claims as amended are patentable over the cited art.

Applicants first note that the Andrew patent has an effective date which is later than the date upon which Applicants invented the subject matter of the present application. Applicants submit a Declaration of Prior Invention herewith demonstrating that the Andrew patent should not be available as a reference against the present application. The Andrew patent provides automatic persistence by reading and saving data from controls. Andrew does not provide control enhancer objects, nor the functionality thereof, as is explicitly called for in each of the currently pending claims. While the Examiner concludes that the Andrew control objects are control enhancers, it is clear

from a reading of the present specification (see: e.g., page 7, lines 2-6) that control enhancers/control enhancer objects are not the same as the controls. Since Andrew is clearly teachings about pointers to controls and not to control enhancers comprising customized interfaces, it cannot be concluded that Andrew obviates the invention as claimed.

Further, Applicants respectfully submit that the Matheny patent does not teach or suggest the system or method as taught and claimed by the present application. The Matheny patent does not teach or suggest control enhancers nor the use thereof in its approach to providing do/redo/undo functionality for document versioning. Applicants respectfully assert that the Examiner's interpretation of the Matheny listed functions (e.g., checkbox 200, play 800, step 802, etc) as controls, as well as interfaces and control enhancers, clearly cannot be sustained by the Matheny teachings. The Matheny functions (or controls) are not control enhancer objects which are customized with base classes and subclasses for specific behaviors for a given window. Based on the present amendments and the fact that Matheny neither teaches nor suggests control enhancers, Applicants respectfully request withdrawal of the rejections based on the Matheny patent.

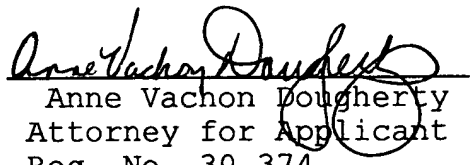
With regard to the Selby patent, Applicants note that Selby does teach a method for rapid GUI development in an object oriented environment. However, the Selby approach is not an object oriented approach to controls. Rather, Selby provides a

structured approach by pre-defining controls and their attributes in a table (500). When a control is needed, Selby pulls up the table and selects the control. Selby does not provide control enhancer objects as interfaces to windows, nor does Selby provide for customization of control enhancer objects with classes and subclasses to provide specific behaviors to be defined for each window control. The present inventive creation and use of control enhancers allows developers the flexibility to provide interfaces (i.e., the control enhancer objects) which do not require specific coding for interfacing with the windows; and, then allows customization of the objects for the required specific behaviors at a base class and subclass level. Clearly the Selby patent disclosure of table-driven control selection is not teaching or even suggesting the creation and use of control enhancer objects as taught and claimed by the present invention. Again, while Selby describes an object oriented environment, Selby's solution is structure driven/table driven. In fact, it can be argued that Selby violates the object oriented principle of encapsulation by requiring a control entity (a central control object) for handling the table data. As such, Selby teaches away from the control enhancer objects of the present invention.

Applicants have amended the claims to highlight the distinctions over the cited art. Specifically, Applicants have amended the claims to recite the control enhancer objects (as clearly supported by the specification).

Based on the foregoing amendments and remarks, Applicants request entry of the amendments, withdrawal of the rejections, and issuance of the claims.

Respectfully submitted,
M. E. Siksa, et al

By: 
Anne Vachon Dougherty
Attorney for Applicant
Reg. No. 30,374

MARKED-UP CLAIMS WITH AMENDMENTS SHOWN

1. A system for providing enhanced functionality for handling each event of at least one event received by a window object having a plurality of window controls comprising:

a plurality of control [enhancers] enhancer objects, each providing an interface to a one specific control for said window object and being customized with specific behaviors from a plurality of base classes and subclasses; and

a list of said control [enhancers] enhancer objects for said window, whereby said window passes an event to all of said control enhancers on said list and wherein said control enhancers determine [determining] which of said plurality of control enhancer objects [enhancers to invoke to] should handle the received event.

2. The system of Claim 1 wherein each of said plurality of control [enhancers] enhancer objects is customized with [comprises] at least one data storage handler.

3. The system of Claim 1 wherein each of said plurality of control [enhancers] enhancer objects is customized with [comprises] at least one data initializer.

4. The system of Claim 1 wherein each of said plurality of control [enhancers] enhancer objects is customized with [comprises] at least one data finalizer.

5. The system of Claim 1 wherein a first one of said window controls is related to at least one second of said window controls, said control enhancer object for said first window control further comprising at least one pointer to the control enhancer object for said second window control; at least one means for determining if an action at said first control enhancer object affects said second control enhancer object; and means for communicating with said second control enhancer object.

6. The system of Claim 1 wherein at least one of said control [enhancers] enhancer objects further comprises means for determining limits to be placed on data related to said control enhancer object.

7. The system of Claim 4 wherein at least one of said control [enhancers] enhancer objects further comprises means for validating data at said data finalizer.

8. The system of Claim 1 wherein at least one of said control [enhancers] enhancer objects further comprises means for identifying data related to the window control of said at least one control enhancer object.

9. A system for providing enhanced functionality for handling each event of at least one event received by a window object having a plurality of window controls comprising:

a plurality of base classes and subclasses representing discrete behaviors;

a plurality of control [enhancers] enhancer objects, each providing an interface to a one specific control for said window object, each of said control [enhancers] enhancer objects being customized with at least one of a plurality of specific behaviors using said plurality of base classes and subclasses comprising at least one data storage handler, at least one data initializer; and at least one data finalizer; and

a list of said control [enhancers] enhancer objects for said window, whereby said window passes an event to all of said control enhancers on said list and wherein said control enhancers determine [determining] which of said plurality of control enhancer objects [enhancers to invoke to] should handle the received event.

10. The system of Claim 9 wherein a first one of said window controls is related to at least one second of said window controls, said control enhancer object for said first window control further comprising at least one pointer to the control enhancer object for said second window control; at least one means for determining if an action at said first control enhancer object affects said second control enhancer object; and means for communicating with said second control enhancer object.

11. The system of Claim 9 wherein at least one of said control [enhancers] enhancer objects further comprises means for determining limits to be placed on data related to said control enhancer object.

12. The system of Claim 9 wherein at least one of said control [enhancers] enhancer objects further comprises means for validating data at said data finalizer.

13. The system of Claim 9 wherein at least one of said control [enhancers] enhancer objects further comprises means for identifying data related to the window control of said at least one control enhancer object.

14. A method for providing enhanced functionality of window controls in response to at least one event received at said

window, said window comprising a plurality of control [enhancers] enhancer objects, comprising the steps of:

receiving an event at said window;

locating at least one interested control enhancer object for said event from said plurality of control [enhancers] enhancer objects;

passing said event to said at least one interested control enhancer object; and

handling said event at said at least one interested control enhancer object.

15. The method of Claim 14 wherein said window comprises a control enhancer object list of events affecting each of said listed control [enhancers] enhancer objects and wherein said locating comprises:

accessing said list of events;

comparing said received event to said list of events; and

determining interested control [enhancers] enhancer objects based on said comparing.

17. A method for rapid graphical user interface development for providing an enhanced control for event handling on a window comprising the steps of:

creating a plurality of base classes and subclasses for discrete behaviors;

creating a control on said window;

instantiating a control enhancer object as an interface to said window for said control;

customizing said control enhancer object by associating selected behaviors to it using said plurality of classes and subclasses; and

passing a pointer for said control to said control enhancer.

18. The method of Claim 17 wherein said associating comprises [further comprising] the steps of:

determining if special data handling is required; and

instantiating at least one data handler if special handling is required; and

assigning said data handler to said control enhancer object.

19. The method of Claim 17 wherein said associating comprises [further comprising] the steps of:

determining if special initialization is required;

instantiating at least one data initializer if special initialization is required; and

assigning said at least one data initializer to said control enhancer object.

20. The method of Claim 18 wherein said associating comprises [further comprising] the steps of:

determining if special initialization is required;

instantiating at least one data initializer if special initialization is required; and

assigning said at least one data initializer to said control enhancer object.

21. The method of Claim 17 wherein said associating comprises [further comprising] the steps of:

determining if special data finalization is required;

instantiating at least one data finalizer if special finalization is required; and

assigning said at least one data finalizer to said control enhancer object.

22. The method of Claim 18 wherein said associating comprises [further comprising] the steps of:

determining if special data finalization is required;

instantiating at least one data finalizer if special finalization is required; and

assigning said at least one data finalizer to said control enhancer object.

23. The method of Claim 19 wherein said associating comprises [further comprising] the steps of:

determining if special data finalization is required;

instantiating at least one data finalizer if special finalization is required; and

assigning said at least one data finalizer to said control enhancer object.

24. The method of Claim 20 wherein said associating comprises [further comprising] the steps of:

determining if special data finalization is required;

instantiating at least one data finalizer if special finalization is required; and

assigning said at least one data finalizer to said control enhancer object.

25. The method of Claim 17 wherein said associating comprises [further comprising] the steps of:

determining if said control has at least one relationship with at least one other control on said window;

instantiating said at least one relationship;

assigning said at least one relationship to said control enhancer object; and

passing a pointer to each of said at least one other control.

28. A system for rapid graphical user interface development for providing enhanced control for event handling on a window comprising:

class means for creating a plurality of base classes and subclasses for discrete behaviors;

control enhancer creation means for instantiating a control enhancer object as an interface to said window for said control; and

control enhancer customizing means for customizing said control enhancer object by associating selected behaviors to it using said plurality of classes and subclasses.

Graphical User Interface Control Enhancers

Mary Ellen Siksa, Steven Pothoven

October 28, 1996

IBM/Advantis Confidential

I. Brief Description

A. Disclosure

A software application design is disclosed to aid in rapid graphical user interface (GUI) development in object-oriented development environments where subclassing controls (buttons, list boxes, edit controls, etc.) provided by the compiler or a GUI toolkit is not appropriate. For example, if a GUI test tool only recognizes native controls for automating testing, and would not work with subclassed controls, then subclassing would be undesirable.

Even though the design was motivated by the environment stated above, it can be used in any object-oriented GUI programming environment.

This design shows how to add additional function to controls for initializing the control with data or other settings, retrieving data from the control, formatting the data, validating the data entered, storing it the control's data in files or databases, handling multiple controls to create the illusion of a single control, and establishing relationships between controls.

The design describes a set of building blocks to use to create GUI windows with sophisticated functions in a very short time.

B. Background

The design is implemented in the project code named Courier, which is the next generation of the IBM Expedite application, an interface to IBM Information Exchange. This product is unannounced as of this writing.

In addition to the authors/designers, the following members of the Courier development team have witnessed and/or assisted in the implementation of the design:

Michael Laskey, Darrell Uthe, Miran Moore, Jason Rendell

The following members of the team are aware of the design:

Carolyn Engelke, James McClure, Mehdi Rakhshani

II. Problem solved

Normally, C++ programmers subclass a set of classes that represent controls included with a compiler or GUI development toolkit or framework. This does not always work, especially if the desired test tool does not recognize subclassed controls. It also leads to a large number of classes subclassed from the same control to provide functions needed. This design results in fewer classes with mix-and-match capabilities.

This design shows how to leave the controls in their native class, yet use a set of building blocks to add functionality to the controls to build a GUI application in less time than it would take to write special code for each control.

Using this design the programmer can develop a set of reusable tools specifically geared towards the special needs of his/her application or market. Existing controls, whether native or enhanced by a compiler or other second party, can participate in the design equally as well.

A. Known prior art

(what does XVT have in their next release?)

While programs like Visual Basic provide controls that are completely functional, they are rarely acceptable for every programming assignment. since these tools are not directly customizable. This design shows ways to enhance or customize those controls as well.

B. Shortcomings

None known.

III. Graphical User Interface Control Enhancer Design

GUI window code, with little exception, is written in an event driven style. An *event* is a mouse click, or a button click, a mouse move, or a key stroke, or any of many other interactions between the user and the program. The Control Enhancer design requires that the GUI windows be written in the event driven style. The basic concept is to pass the events on to the Control Enhancers on the window when they pertain to the controls on the window (a window-size event would be handled by the window itself, while a mouse click on a radio button would be intended for the radio button control).

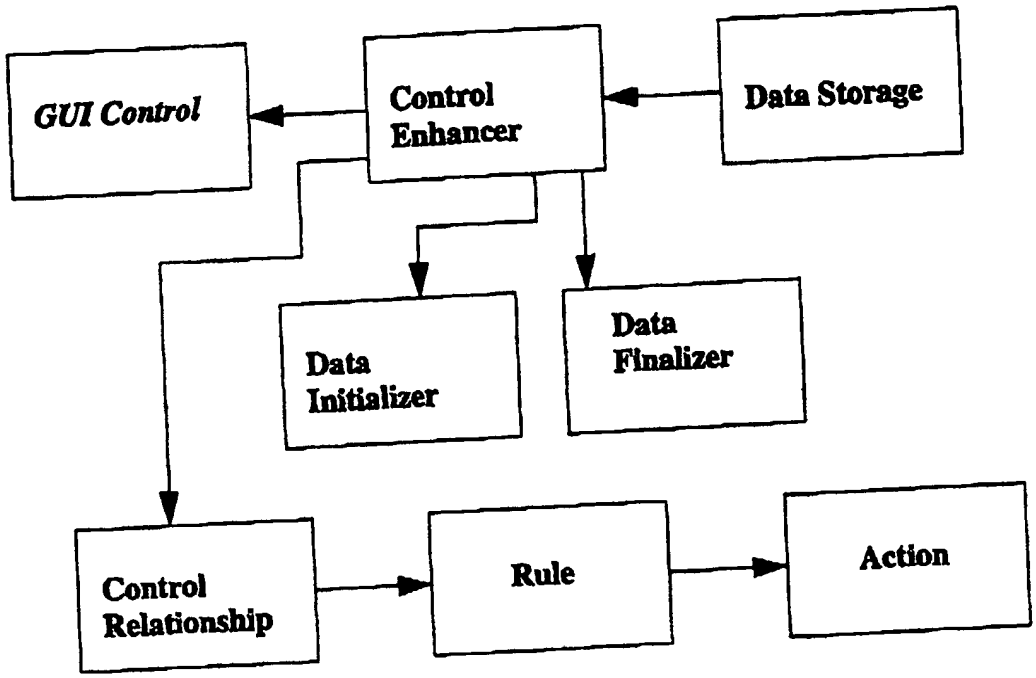
GUI developers are familiar with the model-view-controller design, which works well with Control Enhancers.

The Control Enhancer design consists of these components:

- Control Enhancer
- Multi-Control Enhancer
- Data Storage
- Data Initializer
- Data Finalizer
- Control Relationship
- Rules
- Actions

IV. Overview

A. Diagram



B. Description

Each control on the window is assigned to a Control Enhancer. The Control Enhancer provides a consistent interface to the control for the remaining objects, no matter what kind of control it is (text entry field, list box, combo box, etc.). A different Control Enhancer is designed to handle each kind of control, or a family of closely related controls.

Each GUI window object maintains a collection of Control Enhancers for the controls on the window. Each time an event is received, such as a mouse click or a request to save the data, the event is passed on to each Control Enhancer in the collection. Each Control Enhancer knows whether to handle the event or to ignore the event.

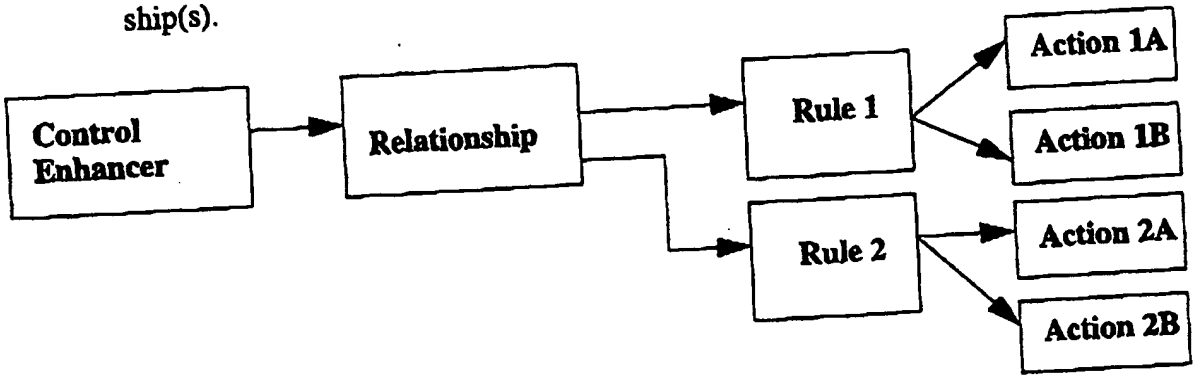
A Data Storage is assigned to each Control Enhancer. The Data Storage identifies where the data should be stored when the "Save" event is received. For example, data may be stored in a database, in an INI file, or someplace else in memory for future processing. A new Data Storage is designed to handle each way that data is stored by the program.

A Data Initializer is assigned to each Control Enhancer to get the initial setting for the control. A new Data Initializer is designed to handle each type of data initialization.

For example, initial data values or settings for a control may come from tables, INI files, or a database. The program may store the data in one format, but needs to show it in another format. The Data Initializer handles any data manipulation that should be done upon initialization. The Control Enhancer activates the Data Initializer when a Initialize event is received.

A Data Finalizer is assigned to each Control Enhancer to prepare the data to be stored by the Data Storage. When the Control Enhancer receives the Save event, it informs the Data Storage, which in turn consults the Data Finalizer to see what data or setting should be stored. A new Data Finalizer is designed to handle each type of data preparation the program needs. For example, the program may need to store a value as 'A', but display it as "All values". The Data Finalizer handles any data manipulation that should be done before the data from the control is stored.

On some windows the programmer will find it necessary to change one control based on the setting of another control. A Control Relationship is assigned to one of the Control Enhancers, known as the Owning Control Enhancer. The Relationship must know about (have a pointer to) the Affected Control Enhancer as well. Each Control Enhancer maintains a set of Control Relationships as necessary. When the Owning Control Enhancer receives a Change event, it passes this event on to its Relationship(s).



Relationships are assigned Rules, and Rules are assigned Actions. Rules are objects that look at the data values of both controls and make a comparison. If the comparison is a match, then the Rule is said to be satisfied. When a Rule is satisfied, the Action is triggered. The Action sends messages to the Affected Control Enhancer to change its behavior.

Only one Relationship is designed, and it will handle any type of Rules. Rules must be defined for each kind of comparison needed between controls. For example, a Greater Than Rule examines the data of the Owning Control Enhancer to see if it is greater than the data of the Affected Control Enhancer. If it is, then its Action is triggered.

Actions must be defined to handle each kind of behavior change needed by a program. For example, when triggered, a Disable Action would tell the Affected Control Enhancer to disable its control.

To clarify, assume we have an Owning Control Enhancer for a checkbox on the window, and the Affected Control Enhancer is managing a text entry field. We want the text entry field to be disabled if the checkbox is selected ("checked"). When the Owning Control Enhancer for the checkbox receives the "Checked" event, it asks its Rule to make the comparison. The Rule is satisfied, and the Action is triggered, which sends the Disable message to the Affected Control Enhancer.

Each control may have relationships with multiple controls on the window. Relationships may have multiple Rules. For each Rule, there may be multiple Actions assigned. The Owning Control Enhancer passes the Change event on to each of its Relationships, the Relationship checks each one of its Rules, and if satisfied, the Rules trigger all of their Actions.

C. Using the Control Enhancer building blocks

Usage of the Control Enhancer consists of assembling the various components in different ways to obtain the results desired. Basically, the assembly of a Control Enhancer works in the following manner:

1. A GUI control is created on the window
2. A Control Enhancer is created and is assigned a pointer to the GUI control, and added to the window's collection of Control Enhancers
3. A Data Storage is assigned to the Enhancer
4. An Initializer is assigned to the Enhancer
5. A Finalizer is assigned to the Enhancer
6. The Control Enhancer is told to Initialize itself
7. If the control has relationships with other controls on the screen
    a. A Relationship is assigned to the Control Enhancer, and is given pointers to both the Owning and Affected Control Enhancers
    b. Rules are assigned to the Relationship
    c. Actions are assigned to the Rules, and are given pointers to both the Owning and Affected Control Enhancers

Once the basic building blocks are completed and tested, they can be combined in a nearly infinite number of ways to handle most any GUI window. Programming a GUI becomes a matter of designing the window and its controls, and then creating the Control Enhancer sets as needed. Only a limited amount of special code needs to be written for the window (when it is so special as to warrant *not* writing Control Enhancers to handle it).

Window classes may be derived from a common base class which provides support for managing a set of Control Enhancers.

D. Enhancing the basic building blocks: Default Finder, Limitation Director, and Validator

When a Control Enhancer is assembled it is assigned a Data Identifier which is used to specify the data for which it is responsible. This identifier is used by all the associated objects. A dummy identifier can be used for controls that do not need to load any initial data or to save the data when done (e.g., a status bar Control Enhancer, which simply displays messages from a file).

Once the Control Enhancer is created, it is told to initialize itself. At this point the Initializer may use the Data Identifier to find any default data through a commonly known Default Finder object, which determines the default data based on the Data Identifier. The programmer needs to set up tables, keyed from the Data Identifier, to provide this information, or to provide the location of the information for the Default Finder so the correct initial data is shown. The most common Initializer is probably the one that displays data from a database field.

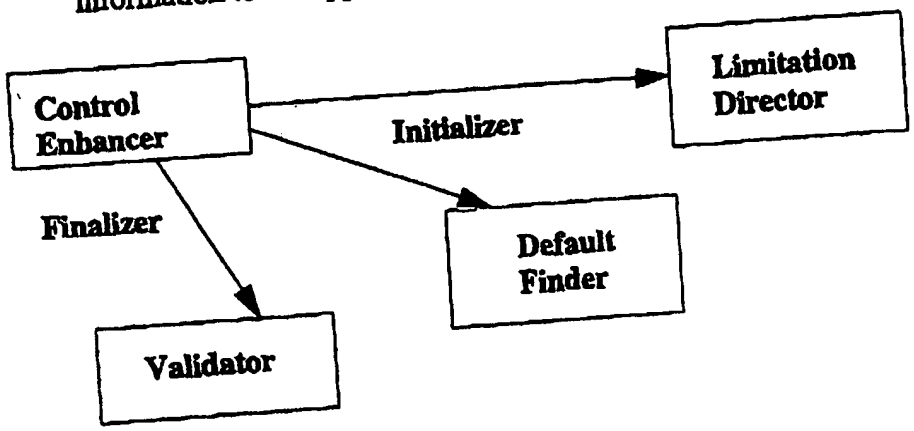
The Control Enhancer initialization code also consults the table-driven Limitation Director, which uses the Data Identifier to find any appropriate limitations to be placed on the data. For example, the maximum and minimum lengths for the data, the maximum and minimum values for the data, and possibly the data type (all character, numeric, date, time, etc.). As is appropriate, the Control Enhancer uses this information to send the messages to its control to adjust its behavior, so that the user is properly guided through using the control.

When the Save event is received, and the data is Finalized, the Control Enhancer creates a Validator object and passes its data and the Data Identifier. The programmer needs to setup tables, keyed on the Data Identifiers, to provide information on if and how the data should be validated. If the Validation object finds the data has been entered incorrectly, it communicates an error message number to the error window, which displays the proper message. Normally, in a GUI environment, the controls restrict the user's selections so that invalid selections cannot be made. However, there are instances where this is not possible, and the Validator becomes handy.

By remembering the Data Identifier for the control with the invalid data, the window code can use this information to find the Control Enhancer for the control that should

receive focus once the error window is closed. This is normal behavior for this predicament, where the user must re-enter data that was not acceptable.

If the data is considered valid, then it is passed to the Data Storage which writes the information to the appropriate place.



E. Multi-Control Enhancers

There are cases in GUI development where the programmer needs to handle a group of controls as a single entity. For example, a date-entry set of fields which consists of a spin button for the year, a drop-down list for the month, and another spin button for the date. Although the date is stored in the data base as a single field, and has only one Data Identifier, it is more likely that the GUI will allow the user to select each component of the date through an easy-to-use control, appropriate for that component.

In this example, we would want to create three Control Enhancers, one for each of the components, but the Data Storage needs to be aware of all three Control Enhancers which is unusual in this design. The answer is to use a Multi-Control Enhancer, which is written to handle the separate components both as a single entity and as its three individual components. The Multi-Control Enhancer provides the same interface as all the other Control Enhancers for its associates, but handles its unique data as appropriate. It may be assigned any Data Storage, Relationships, etc. Usually its Initializer and Finalizer are quite specific to the Multi-Control Enhancer's tasks.

The Multi-Control Enhancer may or may not need to manage the controls with Control Enhancers. It may be easier to manage the controls directly in some cases; however, using Control Enhancers allows the programmer to change the individual date controls later without affecting the Multi-Control Enhancer code. Each Control Enhancer has the same interface with which the Multi-Control Enhancer can work.

A new Multi-Control Enhancer is designed for each case where the programmer needs to satisfy these conditions. These are much more specialized than the Control Enhancers, but found relatively infrequently in GUI development. Once developed and tested, Multi-Control Enhancers become important pieces in the GUI programmer's toolbox.